VALTER GIL
GONÇALVES
CORREIA

Protótipo Modular de um Espelho Inteligente com Interação por Voz

Modular Prototype of a Smart Mirror with Voice Interaction

Universidade de Aveiro
2017

Departamento de Eletrónica,
Telecomunicações e Informática

VALTER GIL GONÇALVES CORREIA

Protótipo Modular de um Espelho Inteligente com Interação por Voz

Modular Prototype of a Smart Mirror with Voice Interaction

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor António José Ribeiro Neves, Professor auxiliar do Departamento de Eletrónica, Telecomunicações e Informática da Universidade de Aveiro, e do Engenheiro Fausto José Oliveira de Carvalho da Altice Labs.

o júri / the jury

presidente / president

Prof. Doutor Augusto Marques Ferreira da Silva
Professor Auxiliar da Universidade de Aveiro

vogais / examiners committee

Prof. Doutor Luís Filipe Pinto de Almeida Teixeira
Professor Auxiliar do Departamento de Engenharia Informática da Faculdade de Engenharia da Universidade do Porto

Prof. Doutor António José Ribeiro Neves
Professor Auxiliar da Universidade de Aveiro

agradecimentos /
acknowledgements

Quero agradecer a toda a minha família por todo o apoio durante este percurso, aos meus amigos por todas as aventuras ao longo destes anos e à Maria por, todos os dias, me incentivar e tornar numa pessoa melhor.
Gostaria também de agradecer ao Professor Doutor António Neves por toda a ajuda, motivação e orientação no decorrer de todo este projecto.
Quero agradecer também às empresas Inova-Ria e Altice Labs por todos os recursos disponibilizados para a conclusão desta dissertação e pela oportunidade de contribuir para um tema que desperta o meu interesse.
Uma palavra de agradecimento a todos os colaboradores destas empresas, que sempre me proporcionaram bons momentos e ajudaram ao longo desta dissertação, mas em especial, um muito obrigado ao Engenheiro Fausto de Carvalho por todas as conversas, trocas de conhecimentos e suporte durante este período.

Palavras Chave internet das coisas, espelho inteligente, sistema modular, interação por voz

Resumo

O mundo está em constante evolução e nos últimos anos tem sido possivel ver um grande aumento de dispositivos electrónicos, presentes no nosso quotidiano, com ligação à internet, a chamada revolução da internet das coisas. Um dispositivo que se enquadra neste grupo e que está a ganhar bastante atenção por parte da comunidade em geral é o espelho inteligente.

Os espelhos inteligentes são espelhos que permitem a uma pessoa ver, para além da sua reflexão, informação relevante com base nos seus interesses. O objectivo desta dissertação é a criação de um sistema que possibilite a visualização desta informação, a interação do utilizador com o sistema e possibilidade da sua customização de acordo com as preferências do utilizador.

O sistema proposto dá vida ao espelho através de módulos que as pessoas podem instalar e configurar de acordo com as suas necessidades. Os vários módulos interagem com o sistema principal, através de filas de mensagens, que são usadas para fornecer informação pronta a ser visualizada e obter do sistema dados providenciados pelo utilizador e pelos sensores existentes.

Para proporcionar interacção com o espelho, foi implementado reconhecimento de comandos usando voz. Foi também criada uma página de configuração onde é possível arrastar e largar módulos para modificação da sua posição assim como alterar parâmetros dos módulos existentes.

Desta dissertação resultou um protótipo de um espelho inteligente modular com interacção por comandos de voz. A sua modularidade permite a criação de novos módulos num grande número de linguagens de programação, podendo o programador escolher a que mais se adequa à funcionalidade implementada pelo módulo.

Foram realizados testes de usabilidade tendo sido obtidos resultados de satisfação que comprovam a qualidade do protótipo desenvolvido.

Keywords internet of things, smart mirror, modular system, voice interaction.

Abstract

The world is in constant evolution and in recent years it is possible to see a great increase in the number of electronic devices, present in our everyday life, connected to the internet, the so called internet of things revolution. A device that fits in this group and that is gathering great acceptance from people in general is the smart mirror.

Smart mirrors are mirrors that allows a person, besides seeing his reflection, to obtain relevant information to his interests. The objective for this thesis is the development of a system that permits the visualization of this information, the interaction of the user with the system and that makes itself capable of being modified and configured based on the users' preferences.

The proposed system brings life to a simple mirror through modules that are available for installation and configuration based on the users' needs. These modules interact with the main system through message queues, that are used both to supply ready-to-be seen information and to obtain data from the user as well as from the existing sensors.

The recognition of voice commands was also developed to allow user interaction with the mirror. Additionally, a configuration page was developed as well, which can be used to modify the position and parameters of the existing modules using a drag and drop approach.

From this thesis resulted a modular smart mirror prototype that can be interacted with using voice commands. The modularity of this system allows the development and integration of new addons in a vast number of programming languages, giving the developer the option and power to choose the language that best suits the module functionality.

Usability tests were conducted and satisfaction results were obtained that confirms the quality of the developed prototype.

Contents

List of Figures

List of Tables

Glossary

Addon Executable code that extends the functionality of our system
AMQP Protocol that defines how messages are organized and sent down the wire
Bing Speech API API for speech recognition and text-to-speech services
C# Multi purpose programming language to build applications that run on the .NET Framework
Chromium Embedded Framework Allows the embedding of Chromium browser in other applications
Configuration Page The portal in our system that allows the configuration of our mirror
CSS Standard language for styling web pages
Dlib Library with useful machine learning algorithms
Express Web framework for Node.js
Face API Online service with face detection and recognition algorithms
FontAwesome A font for icons in CSS
HTML Standard language for developing web pages
IoT A vision of a world where every object can be connected to others and with that, provide information that improves our lives
Javascript Standard language for programming in the browser
JSON A format for exchanging data.
Kinect Motion sensor capable of recognizing depth, voice and gestures.
Main Screen The first screen of our system, where simpler addons appear.
Microsoft Cognitive Services Cloud-based platform with various intelligent services for speech, language, vision and knowledge
Microsoft Speech Platform Is an SDK for developing offline speech applications that run on Windows
MongoDB NoSQL database for documents written in a JSON-like format
Node.js Javascript for server side applications
NoSQL Name given to the group of non-relational databases
NW.js Framework for developing desktop apps using HTML, CSS and Javascript
OAuth2 Protocol for authorization flows.
OpenCV Library with useful functions for image and video processing
Python Multi-purpose programming language
Pug Template engine for the Express framework
RabbitMQ Software for defining message queues that applications can use to communicate with other applications
Raspberry Pi Small and affordable computer that provides easy interaction with the physical hardware
REST A set of guidelines for designing web services that manipulate web resources
RSS A format for delivering updates of online content

SRGS Standard for defining speech recognition grammars
Viewer Screen A new in our system used mainly for complex addons
WebSocket Interface that allows communication between a browser and a server

1

Introduction

This chapter introduces the motivation for this thesis, and a general outline of the remaining chapters.

The past years saw a huge increase in devices connected to the internet, be it, light bulbs, refrigerators, coffee machines, microwaves, air conditioning systems and much more. Nowadays, it is possible to see the rise of a new convenient device, the smart mirror.

A smart mirror works as a normal reflecting mirror while also allowing the display of important and meaningful information on the mirror, while people are getting dressed or brushing their teeth, for example, while also giving it a kind of futuristic look, which may be one of the reasons why people are interested in having one.

Altice Labs [1] is an IT company located in Aveiro that has a long history of cooperation with entities belonging to the National Scientific and Technological System, in particular, the University of Aveiro which has successfully contributed to multiple research projects in the recent years. This cooperation has translated in the application of state of the art knowledge to concrete products that impact the world and solve real problems and needs.

Because Altice Labs is always looking for innovative products and services that can enable a more digital society, and based on the new contexts of contents and services consumption, this thesis was proposed with the intent of exploring and prototyping a new smart mirror system, accessible through voice, gestures and visual interaction using the surface of a mirror to visualize information from both sensors and the internet. This will prove useful to evaluate technological, functional and usage experience characteristics of this type of device and understand if it should belong to the Altice Labs product's portfolio.

1.1 Contributions

The current thesis provides an overview of the smart mirrors being developed and used at the present time, introduces a new modular platform for showing information on the mirror and presents a physical prototype where this system was deployed and tested by several people using voice interaction. It is also presented a configuration page where the user has the tools needed to modify the information visible on screen according to his preferences.

Figure 1.1 shows the proposed system.

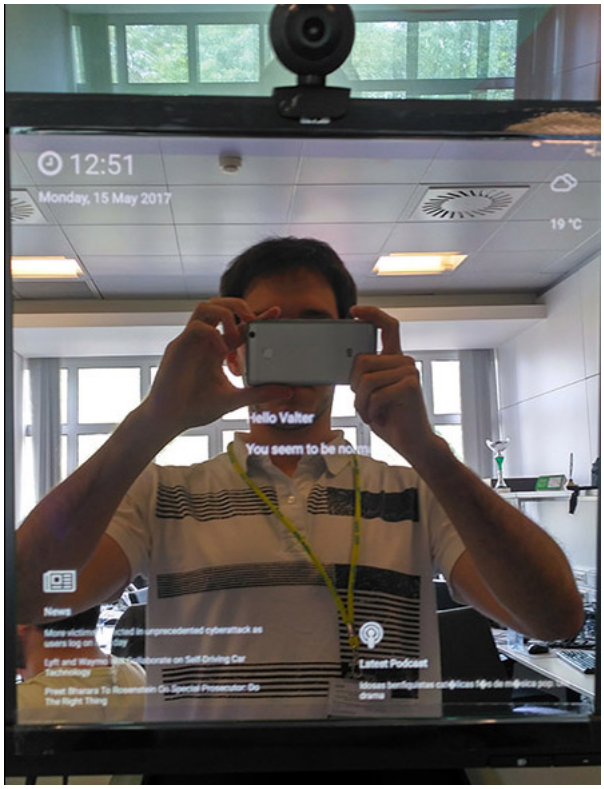


Figure 1.1: The proposed smart mirror prototype

1.2 Structure

Chapter 2 presents some of the existing smart mirrors and their functionalities as well as an analysis of the possible techniques that can be used when interacting with this type of device.

Chapter 3 provides a top view of the architecture implemented during the course of this project

Chapter 4 describes the approaches and technologies used to develop the different pieces of the architecture.

Chapter 5 details the various modules developed to provide capabilities to the smart mirror but also to test the architecture created.

Chapter 6 reveals the experimental testing done to the developed prototype and analyzes the feedback and interactions made by the users.

Chapter 7 exposes the conclusions and problems that originated from the proposed thesis and discusses the future work that may be done in order improve this smart mirror.

2

Fundamentals

Presents the features and technologies being deployed in the smart mirrors of today.

2.1 Existing solutions

Although smart mirrors are now a new gadget that people know relatively well, many attempts were made through the years to create this kind of futuristic device. This section intends to present some of those experiments, that led to the actual state of the art on smart mirrors.

A first effort at creating this new type of gadget was made by the AwareMirror [2] team, where they designed an unobtrusive smart mirror that is able to notify the user about the weather, the next event on his schedule as well as if he should worry about traffic or not. They decided to place a one-way mirror on top of a computer monitor, which blocks the darker colors and allows the information in brighter colors to be seen through the mirror. The mirror uses two infrared sensors to detect the user and a modified toothbrush to identify him, being then capable of showing specific information to that user. Unfortunately, this solution was created in a time where big and cheap LCDs were not around but even with its small size, it is possible to see how cool these features can be.

There is also a smart mirror for ambient home environment [3] intended to provide a "natural interaction between users and the ambient home services". Unlike the previous work, this product is a touchscreen display equipped with a camera that live streams its captures in an attempt to mime a mirror. It has the advantage of being able to display any kind of color and the disadvantage of not feeling a real mirror since it depends on the quality of the camera used. The mirror interface uses face recognition

to authenticate the user and includes widgets developed as Flash containers that enable the user to see the weather, latest news feeds and interact with home appliances such as light switches.

As these type of fun and interactive mirrors gain popularity among the masses, big companies tried to create their own product. As a result, Samsung [4], LG [5] and Toshiba are some of the companies that already presented some smart and interactive mirror products. LG's and Toshibas' don't provide much information but Samsung advertises their smart mirror as having a "polarized film that ensures high transparency (90%) and reflectivity (55%)" and "a proximity sensor that automatically transitions between a mirror and signage". In terms of software, these solutions have support for their respective media content manager which is customizable by the business purchasing it. While these intelligent mirrors from Samsung and LG seems marketed more towards businesses, the Toshiba mirror[6], seen at the Consumer Electronic Show in 2014, seems to be targeting a home environment.

Different from those and unlike the other existing solutions analyzed, the Perseus [7] tries to be a full smart mirror plug and play solution. One of these interactive mirror products, but directed for the home environment and everyday use, which the user can just plug it, switch on and use it.

An image of the product is shown in Figure 2.1, where we can see a person checking his reflection on a Perseus mirror while, at the same time, having access to a digital clock, the current weather, a video playing and the latest news. This product also uses a one-way mirror to create the idea that the information comes directly from the mirror (as opposed to a monitor) and comes equipped with a 2GHz quad-core processor, 2GB of RAM as well as a high definition camera, speakers, Wi-Fi and Bluetooth connectivity. It also seems to provide extensibility through apps, that let you know the weather for the day, read the news, check your commute, watch videos, read your text messages and receive notifications for your calendar events. In the presentation video[8] is also announced the possibility for voice interaction with the mirror's system.

When it comes to software, the Perseus Mirrors company does not provide the source code, nor gave information about the operating system that powers the Perseus mirror, and so it is not possible to understand all the capabilities of the system or even if it's possible to develop applications or more features for it, but the Perseus makers announced the release of a developer SDK in the future.

At the time of writing of this thesis, most of the smart mirrors found and shared on the internet are built as a side project and/or hobby. Michael Teeuw was one of these "makers" who created a smart mirror for himself and shared it with the world. To make the mirror useful and intelligent he developed an application with a set of modules to allow user interaction. That piece of software is now called MagicMirror^2 [9] and one

Figure 2.1: Perseus Mirror.

of the main characteristics of this system is that it is built around a modular design. That means that it is extensible and anybody can implement a module with a new and desired feature for the system. The creator decided to open source this system from the start and since its birth, it has received a lot contributions and as a result, it has a wide variety of modules available. The main and out-of-the-box features found in this software are: Clock, Calendar, Weather, News, Compliments and Alerts.

As with MagicMirror[2], many of the existent solutions are open source and that allows us to have a more detailed view of the software systems that run on present smart mirrors and understand how theses systems are developed.

The mentioned software was developed using the Electron framework, which allows the development of web applications using the standard HTML5, Javascript and CSS languages running in a joint Chromium and Node.js process. By using these technologies it can easily run in the three main computer platforms (Windows, Linux and macOS) and makes it also possible to be powered by using a full-sized desktop, a laptop or just a simpler mini computer like the Raspberry Pi.

Figure 2.2 tries to illustrate the main architecture of this system. The system invokes a main view (which consists of a HTML web page being rendered on a Chromium process) where the different modules are loaded and shown to the user. When a developer builds an extension to this system, he programs the logic and enumerates the necessary stylesheets and dependencies. When the mirror is turned on, a module loader component is initiated and for each module present, imports them and their required

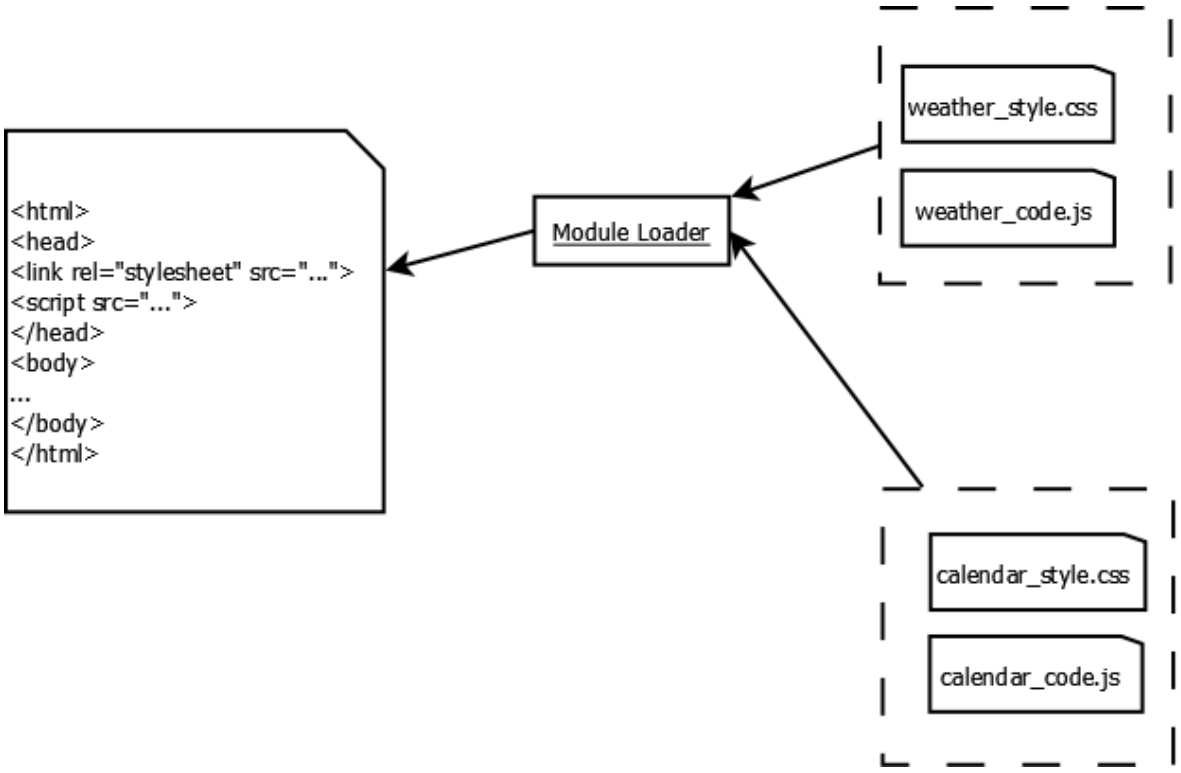


Figure 2.2: MagicMirror^2 module loader architecture.

dependencies by adding the file as a stylesheet link or script to the main HTML page.

When talking about smart mirror projects that were done for fun and as a hobby we must mention the HomeMirror [10]. Figure 2.3 represents the author of the project taking a photo of the built mirror and system (mounted on what seems to be a kitchen wall). We can see the system showing common features like the digital clock and weather and also a custom feature that advises when to water plants. It intends to be another information system for smart mirrors and was developed as an application for Android devices. From looking at his source code we can conclude that the software was not designed with an extensibility pattern in mind and comes with a set of predefined features. It allows the user to check the time and weather and advises the user if he should bike or not based on that weather information, data about a particular company stock market, the person's next calendar event, the latest "XKCD" comic and emotion detection based on facial recognition.

The code is open source and available online. In the opposite to the MagicMirror^2 and because this application wasn't developed based on a modular design, in order to add new features the developer must create a new function with the desired logic and then modify the main view and function to display its information.

More recently, another effort at creating a middleware where people can develop applications for a smart mirror was made [11]. The designed software uses a web

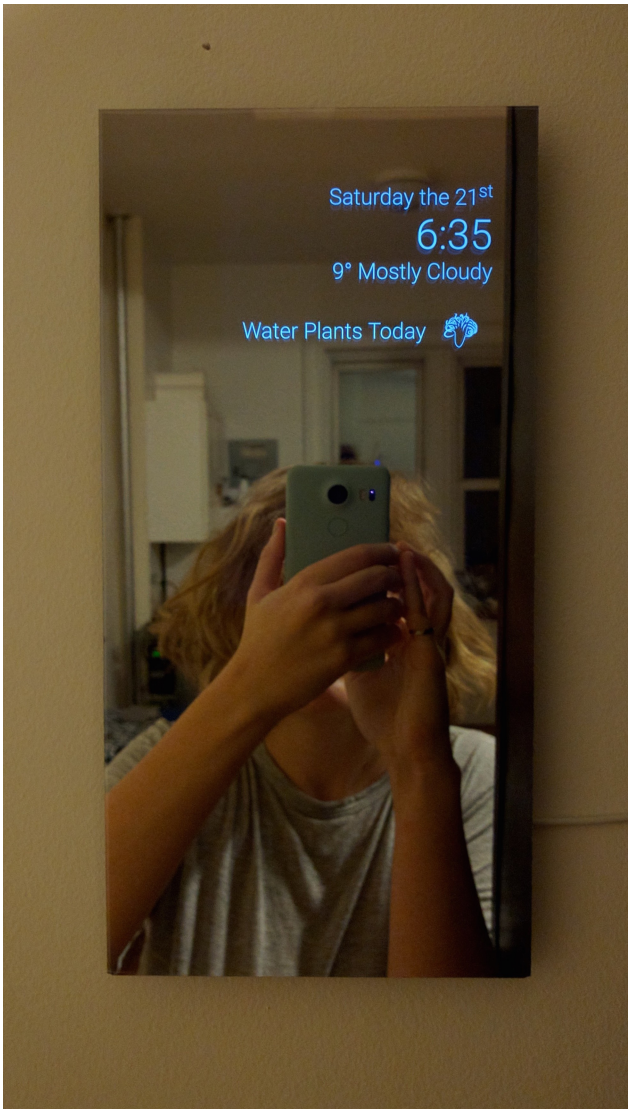


Figure 2.3: HomeMirror

browser to display the data obtained from the plugins. These plugins communicate with the main system (running on the browser) through WebSockets and can be created using HTML, CSS and Javascript or another language when Javascript limitations are too much. The platform implements a modular design pattern and the plugins are enabled or disabled by storing or removing them, respectively, from the specific plugin main folder. The developer chose to differentiate the plugins based on how much they communicate with the main system and so, they are divided into three categories: one shot plugins that are only run once while the system is on, periodic plugins which are scheduled to run from time to time and persistent plugins that connect to the server and are executed until a stop order from the server.

To facilitate the management of the various plugins, the author also developed a web interface where the user can manage the plugins by enabling and disabling them,

change their configurations and their position on the screen.

The author constructed a smart mirror like the ones we can found nowadays in the DIY hobbyist community with a one-way mirror on top of a computer monitor secured by a wood frame. Different from the usual, is the fact that she added a touchscreen module on top of the Raspberry Pi (the platform chosen to run the system) to make possible the interaction with the smart mirror through touch. Using a custom plugin, that touchscreen allows for the user to write notes and save them on the mirror display.

While already developing our own prototype and system software, a new competitor arrived in the field. Its called Duo[12] and its design appears much more polished than the previously analyzed solutions. From Figure 2.4 and the other images available on the product website we can see the normal addons we also find on other smart mirrors, weather information, a digital clock, updated news and to-do list. In addition to those, figure 2.4 also gives us a glimpse of a video and music player as well as integration with a type of air conditioning system.

Despite being only in a pre-order state, the website for this product says that we can expect a full HD 27-inch display, enabled with multi-touch up to five fingers and also controlled using voice commands. The Duo mirror also incorporates a personal assistant, named Albert, that seem to act like the already common, Siri and Cortana. The user has the capability to customize several traits of this personal assistant, such as the assistant's name, his gender, his accent and his jokes.

From the website, we can also deduce that the software controlling this mirror will be also based on a modular architecture with different applications, giving the user the possibility to download more from an App Store. Aside from the more usual apps, it is interesting to see the possible interaction with IoT systems like in this case, the room lights or the air conditioning system.

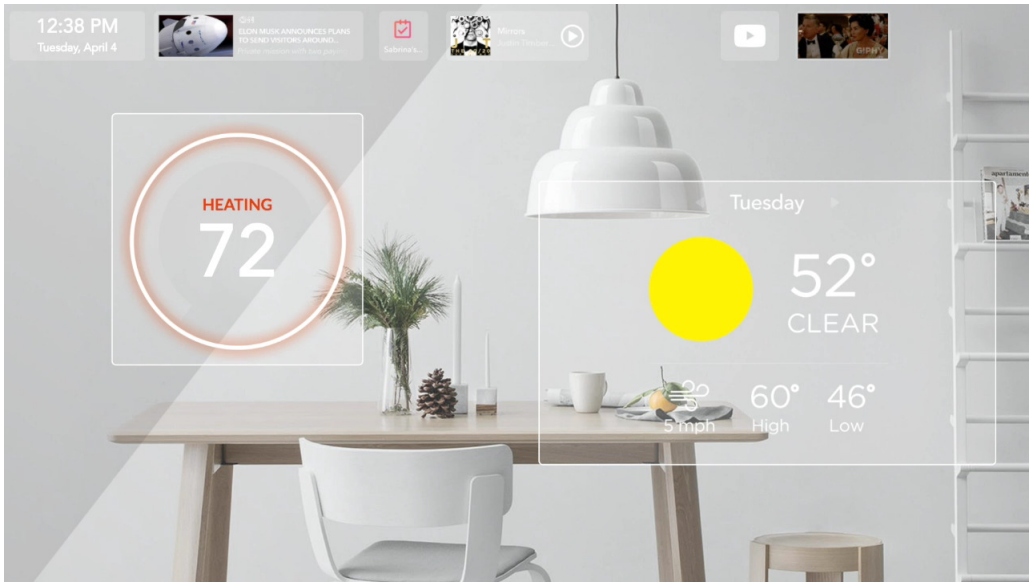


Figure 2.4: Duo Mirror

2.2 Interaction with the user

Designing the correct user interface for a new gadget is an important task because it can mean an easier or harder adoption and for that we have to consider the way the mirror will interact with the user and in the opposite direction, the user interaction with the mirror.

Seeing that our smart mirror will be hanging on a bathroom, wardrobe or any other house wall, we are not really interested in using a mouse or a keyboard for interaction, as that would mean having cables lying around and a surface where to use them which is not always the case, and also it would take await the futuristic look from the mirror. A "search" for the best interaction mechanism was needed, and this section describes some of the options available to interact with a smart mirror and receive some feedback from it.

2.2.1 Touch

One of the possible ways of user interaction with the mirror is through a touchscreen which is, at the moment, a common way of interacting with elements in a display [13] [14] [15] . This technology could be implemented by opting for a touchscreen foil and then applying it to the desired display, or using a monitor with built-in touchscreen. Normally, bathroom or wardrobe mirrors are quite big and so, the above options become quite expensive for displays with our desired sizes. Another downside of using a vertical touchscreen display is that it becomes physically more demanding to use [16]. Finally, using a touchscreen in a bathroom environment might not be a good idea because of the humid hands.

A distinct manner to interact with the mirror display would be to use a mobile phone with touchscreen [17] [18] and a specifically developed smart mirror application, where clicking on the mobile surface would trigger a corresponding alteration to the smart mirror state.

2.2.2 Vision

Although having one camera in a bathroom may lead to privacy issues, they are taken into consideration by the interesting and useful features that can be developed with it.

A separate solution from using touch would be to interact with the proposed mirror using everyday cameras and vision algorithms to recognize hand gestures, which is a technology that was made very accessible, to a vast amount of people, thanks to products like the Kinect. This would be possible to implement in the proposed solution, as today's cameras provide a good enough image resolution and are cheap. (Libraries for hand and gestures recognition are also easy to find.)

When interacting with this kind of intelligent device, a person expects the device to know and realize, at least partially, who is he interacting with. For the proposed smart mirror to provide a meaningful connection and interaction with its user, there is the need for the mirror to recognize the user and extract meaningful information about him. In our case, this meaningful information is the emotional state of the person.

Face recognition, the method of detecting and verifying a person's face in an image or video, has come a long way and is now possible to add face recognition to any application by simply calling a cloud service. There are many companies and startups developing this kind of services but the most known might be Microsoft Cognitive Services and Google Cloud Platform. Karios[19] and Face++[20] are also some of the startups developing products in this area. Using these services speeds up development of the application while having cross-platform availability and means less worries about importing and learning how to use a new library. The developer has is also trying to use the state of the art recognition technologies. Table 2.1 compares some of the available cloud solutions.

Google Cloud Vision API falls under the Google Cloud services related to the use of computer vision algorithms for things like face recognition, detection of labels, text, landmarks and knowledge extraction from images, detecting sexual or violent content in photos as well as evaluate the emotion from the person's face.

Kairos[19] is a "Human Analytics platform designed for innovative, data driven businesses". They also produced software capable of detecting faces in images, evaluate the similarity of two faces and supply data about those faces like emotion, gender and age.

Service	Emotion Detection	Quota (monthly)
Face++	Smiling	Unlimited
Kairos	Joy, Surprise, Sadness, Anger, Disgust, Fear	5K* (only during trial)
Google Cloud Platform	Joy, Sorrow, Surprise, Anger	1K calls
Microsoft Cognitive Services	Anger, Contempt, Disgust, Fear, Happiness, Neutral, Sadness, Surprise	30K calls

Table 2.1: Comparison of Face Recognition capabilities offered by the different services

Face++[20] is yet another of these platforms that is able to detect faces within an image, matching those faces with previously saved ones. In terms of analyzing a photo for facial attributes, it only tells the confidence index for "is the person smiling?".

Microsoft Cognitive Services is another contender and under the "Vision" category, provides services that let us analyze an image, understand and categorize its contents. One of the service from this section, named "Emotion API", can be used to "analyze faces to detect a range of feelings and personalize your app's responses" [21]. By analyzing an image consisting of some person's faces, this service returns, for each face identified, a list of emotions (anger, contempt, disgust, fear, happiness, neutral, sadness, and surprise) each with a confidence level.

The smart mirror also tries to identify the user in front and that aptitude relies on the "Face API" service which makes possible to "Detect human faces and compare similar ones, organize people into groups according to visual similarity, and identify previously tagged people in images" [21]. This API is used to verify the likelihood of two faces returning a confidence value of that likelihood.

Kairos claims it can detect six different type of emotions, but simple tests made by us with their demonstration application always revealed bad results and the free plan offered with five thousand calls to the API can be only used during the trial period which only lasts thirty days. Unlike Kairos, Face++ offers unlimited calls to their API but has the disadvantage that only detects if people are smiling (and how much) and no more emotions. Microsoft Cognitive Services was the chosen platform mainly because of the higher free usage quota that they grant but also because, when compared with Google platform, they detect a bigger number of emotions and our tests showed good results. Using the "Face API" service, we compare a photo of the actual user with one stored in the database and based on the result we know if the mirror is interacting with its main user. That allows us to show content based on that specific user configuration. This should not be used as real security measure but it is enough for this application

since all we want to do is have some confidence that we are interacting with a certain user.

It was decided that the developed mirror system should have some functionalities that would enable a user to interact with the mirror in some fun and useful ways. One of them is the ability to try out new fashion accessories like glasses, ties or hats. A normal approach to this problem is to extract key points from a person's face such as the position of both eyes, nose and mouth and then place the desired content over the person's face. To that effect three options were analyzed. One of the options tested was the library DLib. Quoted from the official documentation, "Dlib's face detector is made using the classic Histogram of Oriented Gradients (HOG) feature combined with a linear classifier, an image pyramid, and sliding window detection scheme." [22] The CLandMark framework was also tested. This library is an open source facial landmark detector developed in C++. Dlib was the chosen solution mainly because it is a well documented API and easy use. When compared to the CLandMark framework, Dlib performed better, showing quicker results. Adopting a cloud based solution was not considered because we would need to send every obtained frame to the online server and based on its response, modify the image. This type of implementation just doesn't work for a real time application.

2.2.3 Speech

Personal assistants like Apple's Siri, Microsoft's Cortana or Amazon's Alexa are becoming more and more mainstream and all of them use voice recognition and voice synthesis to interact with users. As a result people are used to this technology, which takes this technology as a good option to use in our project.

Between letting the user interact with the mirror through touch, using a touchscreen display, or hand gestures, with the use of cameras and vision algorithms to recognize them, or voice, the solution chosen for this project was the latest because, if we think in the context of a bathroom, most of the time, a person already has a schedule to follow and doesn't want to be waiting for his gestures to be processed when he could be using that time to clean his hands or face, and interacting with a touchscreen while having wet hands is a bad idea.

There are various ways to implement voice interaction in a system. Both Windows and macOS provide an application framework interface (API) for Speech Recognition, but by using it, we are restricting our software to running on a specific platform. The CMU Sphinx [23] speech recognition toolkit solves that problem and is open source. Another solution passes by using a cloud service like the one mentioned in the face recognition part.

Some of these services are Microsoft Cognitive Services, IBM Watson Services and

Google Cloud Platform and all of them provide a Speech-To-Text API which allows us to recognize what a person is saying in front of the mirror and provide this type of interaction. Table 2.2 compares the Speech-To-Text options available from the cloud. From it, we can conclude that the Google service allowing a wide range of audio formats and sample rates but provides the lowest time of free usage. The Watson platform also supports the main audio formats at 8 KHz or 16 KHz, free below one thousand minutes. Microsoft service accepts only the WAV audio format with 3 possible codecs at 16 KHz and the free plan allows up to 5000 requests to the server which can mean 417 minutes at an average of 5 seconds duration for each sound sent. Therefore, the Google Cloud Platform didn't strike us as the best option for this project but the other two options were equally good. The proposed solution uses the Microsoft Cognitive service, in part due to other APIs from this platform being seen as the best choice (and also because it was easy to use.)

Service	Audio formats	Sample Rates (KHz)	Quota (monthly)
IBM Watson Services	FLAC, PCM, WAV, OGG, mu-law	8 or 16	1000 min
Google Cloud Platform	Linear PCM, FLAC, AMR Narrowband and Wideband	Between 8 and 48. Optimal is 16	60 min
Microsoft Cognitive Services	WAV using PCM, Siren or SirenSR	16	5000 requests

Table 2.2: Comparison of Speech-To-Text capabilities offered by the different services

These were the studied options when talking about how a person can interact with the considered mirror and how the smart mirror can better interact and present content to it's user.

Given these points, we conclude that developing the smart mirror application using a modular design pattern leads to a more customized experience to the user, in that every module represents a mirror feature that can be easily coupled or decoupled from the interface and facilitates the improvement and development of new features and extensions because the developer only needs to create his module using the known APIs and functions available. From these projects it is also possible to understand some of the more needed and common features that people expect to have in a smart mirror. Those are commonly a digital clock, a calendar with events notifications, news feed...

Based on the knowledge resulting from the analysis presented in this chapter we developed a modular prototype of a smart mirror. In the next chapter we explain how

the interface system is going to be designed to better implement all these concepts and features.

3

Architecture

In this chapter it is presented the decisions made about the architecture was going to look like and the decisions that led to the architecture.

Having studied the previous work related to this project, one of the main features used by other mirrors [7][9][12] that seemed really important to implement in our solution was the concept of extending the capabilities of the software through applications. This is possible using the modular design pattern, in which the system code is isolated and divided in meaningful pieces, which are commonly addressed as modules and are in charge of one or a group of similar features.

Adding or removing a module from the system should not create any problems for the other modules or the main application, and so it allows users to remove and change features in the mirror easily. On the other hand, developers also have an easy development of addons because they only have to worry about how to develop their intended feature and not so much if the development of their module is going to interfere with other addons.

The usual approach to design a modular system, is to create an interface with the functions that every plugin has to implement. The main system should then have a configuration file which lists the enabled addons. Parsing this file should give enough information to the system for it to able to create an instance of the plugin and call its functions (for example run()).

It allows for the abstraction of their code but it also means that it has to be used with the same programming language used to create the main system. To avoid this, we propose a modular architecture based on message queues. This means that the different addons are not forced to implement any mandatory function and instead are

only required to communicate the result using the message queues designated for that purpose. The block diagram for the proposed solution is presented on Figure 3.1.

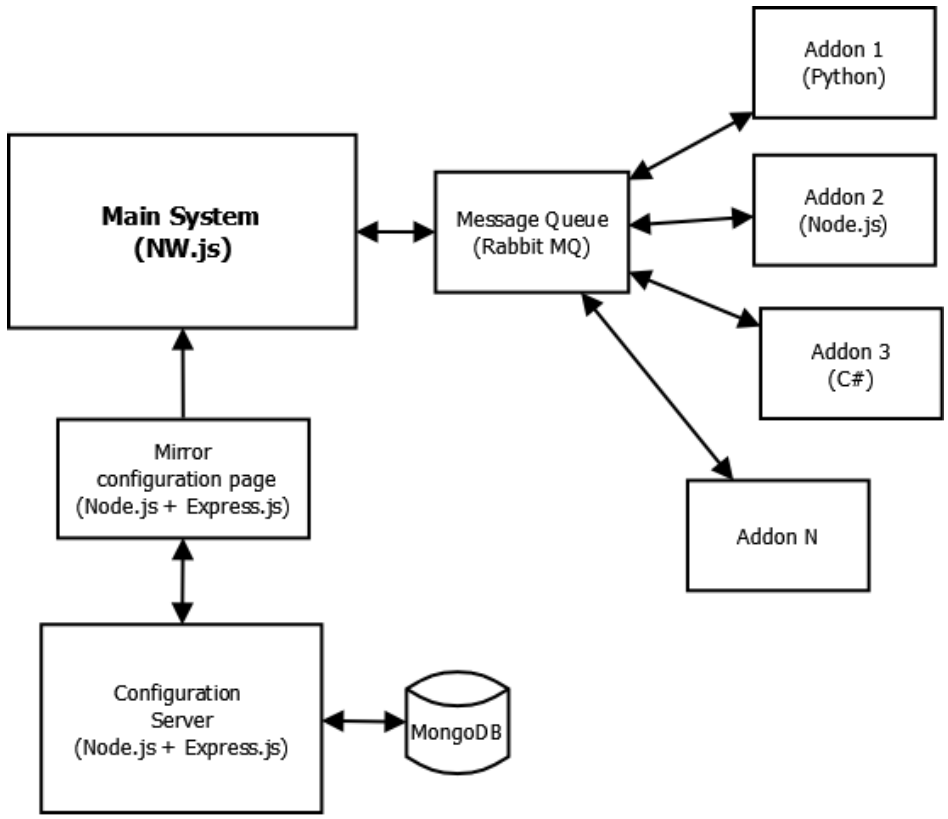


Figure 3.1: The proposed solution architecture.

Each addon has a configuration associated with it and that enables the user of the smart mirror to customize it based on the users interests. To simplify the configuration part of the process of installing a new mirror in the house, the software running on the mirror provides access to a configuration page. This configuration page connects to an online server (eventually property of the maker of these smart mirrors), also created for this solution, where every user's configuration is stored inside a document-based database.

With the existence of an always-on server we have a safeguard of the device's and user's configuration in case of an hardware failure but also allows for a quick configuration when the user decides to buy a new device. For now, no mobile apps were developed, designed to modify the configurations without being connected to the same network of the mirror, but having the configurations available on a cloud server that could become easily a reality.

The configuration page uses this server to obtain all the existing addons possibly usable in the smart mirror but also the current configuration for the logged in user.

Once the user saves his desired configuration, the new addon structure and disposition

will be downloaded from the configuration server and stored in the main system directory inside an addons folder, once the mirror is switched on. All the addons installed in the system can then access these local configurations without the need to access the configuration server which means a quicker startup time for the addon.

As in other programming languages, Node.js, makes it possible to import new capabilities from other single files or full modules using the "require" keyword. An initial experiment made with the intent of implementing a modular architecture was to create a "main.js" file, representing the main application and where our methods for printing information to the screen would be available, and two files, "weather.js" and "clock.js", which would be our first addons, allowing the user to check the current weather and time. These last two files would do the heavy work and have a known function that could be called by the main application to obtain the final output. The main application would then be able to print the returned information on the screen. A visual representation of this architecture can be seen in Figure 3.2.

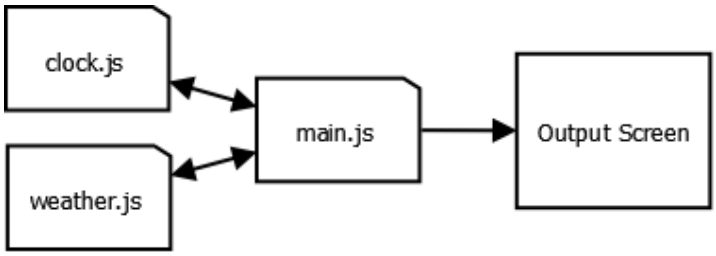


Figure 3.2: First implementation of a modular architecture

For simple extensions, this type of architecture design worked, but once the need to work with sensors, like a webcam or a microphone, arrived, Node.js didn't possess, at the time of development, libraries that allowed for better operations when working with video and data obtained from the webcam. One of those libraries is OpenCV[24], which is a known library, with many functions useful for image and video processing. OpenCV is available in C, C++, Java and Python but is not available as a Node.js library. Although Node.js makes it is possible to load C or C++ code as if they were normal Node.js modules, it is, currently, a complicated process and it involves modifying the existing code in C or C++ to code complaint to the Node.js Addons guidelines.

After that would still be needed to interact with the created module through Node.js as it would be necessary with any other library. The proposed project took another approach. We decided to have modules created in almost any programming languages and let the developer choose the one he prefers to develop his addon, and then communicate needed information using message queues.

The chosen framework in charge of directing our messages between the main application and the different addons is RabbitMQ and it implements the Advanced Messaging

Queuing Protocol. The AMQP protocol details how the data should be organized and transmitted over the network and uses the concept of Exchange, Route and Queue. Queues are a common data structure in which the first object placed is also the first object to leave the structure and allow the trading of messages between to entities. Messages sent to an exchange are routed to the different queues connected to it based on different types of criteria. Our project uses a Fan-Out Exchange, where any queue connected to the exchange receives a copy of the message published to the exchange. Since AMQP is a wire-level protocol with public specifications it is possible to create libraries that implement those specifications in a great number of languages and RabbitMQ already provides many of those implementations as well as good documentation on how to create queues and exchanges along with sending and receiving messages. Those reasons allow for the development of new addons and extensions in any language capable of interacting with RabbitMQ or the AMQP protocol.

3.1 Hardware

In terms of hardware and physical materials, our smart mirror follows the same steps of many smart mirrors that came before it. The most important piece needed to create a smart mirror is the mirror. Previous smart mirrors use a one-way mirror in front of a monitor to give the idea that the information is really coming from the mirror itself. The proposed solution, drawn from that knowledge, uses the same mechanism.

The most notible differences in current smart mirrors arise mainly from the size of the screen displays and the computers powering them. For our prototype, Altice Labs provided an LCD display with 22 inches of diagonal and a one-way mirror with a very good reflection coefficient. At the present, and because this smart mirror is still in development and not yet ready for market and mass consumption, we gave little interest to the frame around the mirror and we aren't using any around the mirror. We are also, only using the one-way mirror taped to the monitor and using a 14-inch laptop to power the main system. This laptop has the following specifications: Intel i5, 4GB RAM and 256GB HDD. In addition to these components we also added an external webcam, model Microsoft LifeCam VX-500, to the smart mirror, which gives us the freedom to experiment and find the best place around mirror where we can put the webcam. Currently we placed this webcam in the upper side of the monitor and since it has a incorporated microphone, it allows us to talk directly into the mirror, which improved our results when using voice recognition, mainly, when compared to the laptop's microphone.

A more completed product is already being developed, which consists of a metal frame with all the necessary components together in a box. This new version of the

prototype will have a 32-inch LCD display, one MSI CUBI 2, which is a mini computer with an Intel Core i5-7200U and 8GB RAM, in conjunction with a webcam, all in the same frame.

4

Implementation

In this chapter it is described the implementation details of the main features of the architecture.

4.1 Register a new account

The first time a user turns on the smart mirror, it will be in a blank state and won't have knowledge of any user accounts, so the default configuration will be used. The default configuration consists of a weather addon, located at the top left of the screen, with information of the weather in Aveiro by default and a clock addon located at the top right, as we can see in figure 4.1.

Figure 4.1: The mirror screen with the default configuration

To customize his mirror screen, a new user has access to a configuration page which, at the beginning, presents a login and registration form. As seen in figure 4.2, to register a new user account for the smart mirror, the user should provide a name for the smart mirror to associate the user with, an email that will be used by the system as the user identifier and will allow the user to log into the configuration page, the desired password and a photo of the user's face that will be used to identify the user when he appears in front of the mirror.

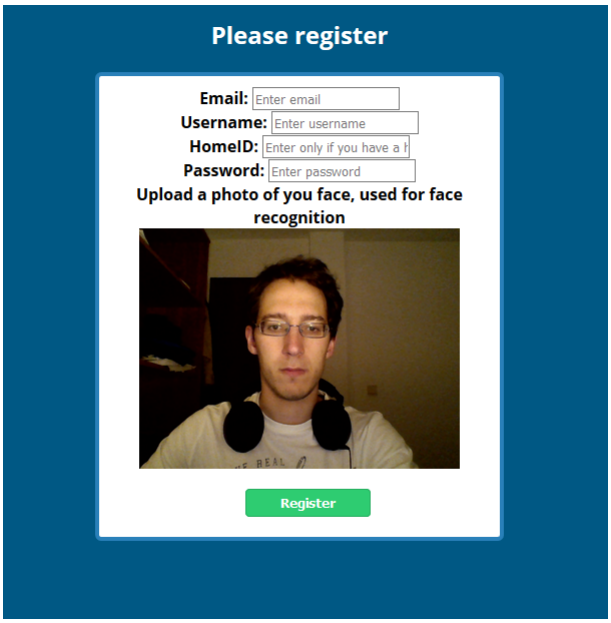


Figure 4.2: Registration form for a new account

Using face recognition to detect the user provides the basis for a context-aware mirror that can adapt it's screen and the information shown based on the identified user. The proposed system uses the Face Identification offered by the Microsoft Cognitive Services. To use this capability, first the system has to send some information about the user to the those services. With regard to this service, the developer has to understand the four different entities used by the service and how they are related to each other. Those are:

- **Person** provide access to a group of actions used to manage a person. Examples of actions used by our system are: "Create a Person", "Get a Person" and "Add a Person Face"
- **Person Group** comprises operations to handle Person Groups objects, which are a group of the above Person objects. Examples of operations used by our system are: "Create a Person Group" and "Train Person Group".
- **Face** is a collection of services possible of being requested by the developer to get meaningful information out of the data inserted in the Face API. Examples of services used by our application are: "Detect" and "Identify".

- **Face List** allows the management of list of faces. Services from this category are not used by the developed system.

After the user provides the information specified at the start of this section and hits "Register", that same information is sent to the /**createUser** endpoint. Figure 4.3 displays the workflow followed by the service behind this endpoint and is described below:

1. In case it is the first registration on the house, starts by creating a Person Group entity using "Create a Person Group" with the id associated with the house where the smart mirror is located.
2. Uses the "Create a Person" service to create a Person inside the previously created Person Group. To use this, the name of the person is a mandatory field and is set to the name given by the user and an optional field is the userData which can be used to store any useful data necessary to the application such as: person's age, gender, hobbies. The developed solution uses this last field to save the user email which works as the user identifier inside the main application. The data stored in this field is limited to 16KB.
3. After having a person object created, we grab the photo sent by the user and use "Add a Person Face" to link it to the Person entity created in the step before.
4. The Person Group is trained using "Train Person Group". After this step the application can now use the "Identify" service to recognize the person in front of the mirror.
5. If the previous steps without any error, the name, email and password is saved in the configuration server's database.

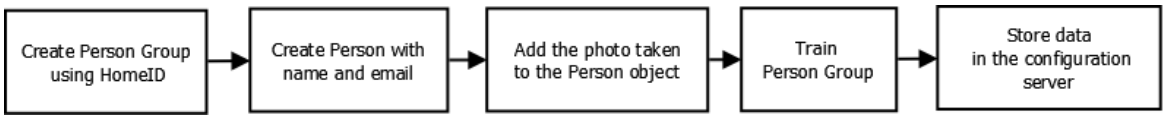


Figure 4.3: Account registration workflow.

Although we use an online service for face recognition makes it possible to have a smaller processing unit which manifests its advantage in the lower cost of production of such a device and overall a smaller size and footprint in the mirror itself, it also has the disadvantage of making the mirror unusable when there is not an internet connection. Thankfully, due to the system architecture and the way it is developed, a developer familiar with the concepts and algorithms for face recognition can create an offline solution and use it as the way of identifying the user in front of the mirror, instead of the online version created for this system.

4.2 Configuration page

Upon creating an account, the user can now be recognized by our smart mirror but first, he should add some modules and applications. This process also happens through the configuration page once the user logs in.

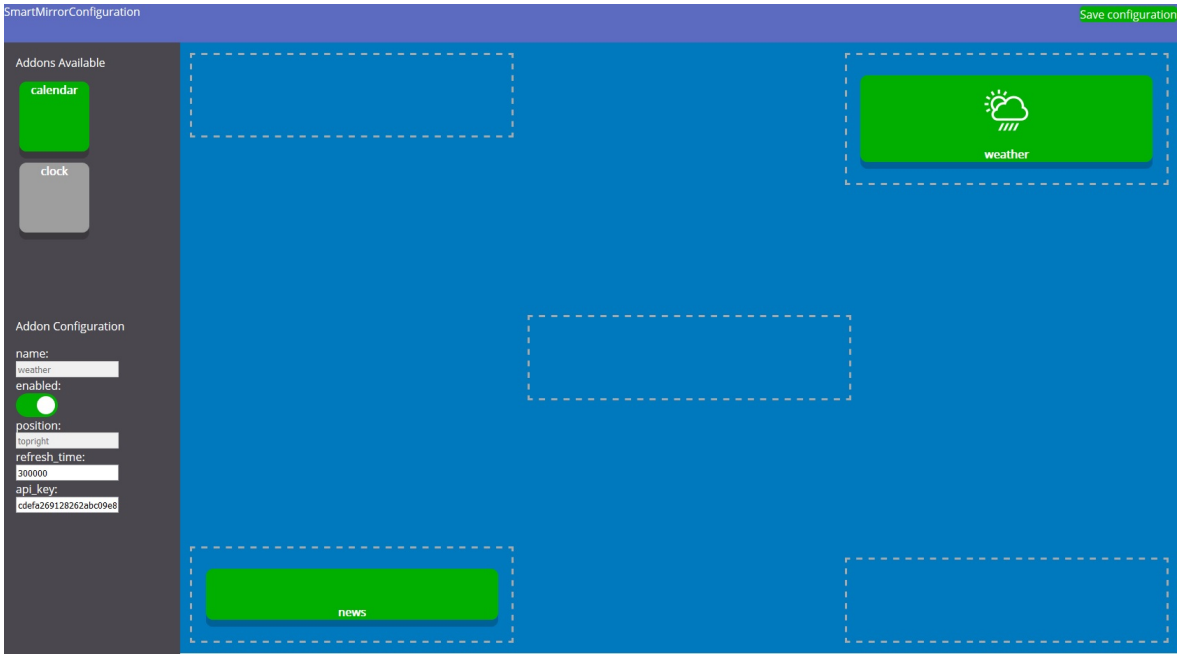


Figure 4.4: Configuration page.

The configuration page was developed using Node.js and Express. Node.js is a cross-platform runtime environment built on the Google's V8 Javascript engine with the purpose of developing server-side applications using the Javascript language. Express is well known web framework for Node.js with support for routing and template engines. Templates engines allow the developer to create a template file for a web page that will contain both static and dynamic information. During the webpage rendering, the template engine will be in charge of binding the values given by the application to the template. From the various options and supported template engines, "Pug" was the chosen one to create the configuration page. We have a template for the login page and a template for the configuration page. The login template holds the information of how both the login and register form will be shown. And both of them consist of a form with some input fields on it. The login page is the page seen by default but using javascript we show the one the user is interested in and hide the other one.

After the user correctly logs in, his current configuration page is presented. This page is divided in three sections, a header, a sidebar on the left, and the main area, that uses the rest of the available space. The header is used to display the logged in user and presents a "Save configuration" button. On the left, the sidebar presents all

the available addons possible of being added to the smart mirror. If the user clicks an addon, information about the configuration for that specific addon will be shown. For example: its name, its position on the screen, if it is enabled, etc. The main page reflects the present state of the smart mirror screen and like the main screen in the mirror, it is divided in five different positions, top left, top right, bottom left, bottom right and center. And easily the user can drag an addon, either from the "Available Addons" section or from the already placed addons on one of the five position and drop it in the new desired position on the screen. Most of the addons can then have configurations specific to them, like API keys or for example, in the case of a weather addon, the city for which we want to see the weather information.

As soon as the configuration are saved and stored onto the configuration servers, the developed software downloads a zip with the latest version of the addon and it's configurations. Those are stored in local storage under the specific addons and user folder, so different users can have different addon configurations. Each addon consists of at least one source code file (or binary) and a configuration file named "conf.json", stored in JSON format, along with other resource files needed to the addon. Code 1 represents a sample configuration file.

```
{
 "name":"news",
 "enabled":"true",
 "position":"bottomleft",
 "refresh_time":"1800000",
 "api_key":"6d2d6e3f8cb14494899a0d6657aac870",
 "news_number":"3"
 }
```

Code 1: Sample addon configuration file.

From code 1 we can observe that the schema used consists of a simple key-value pair composed of the configurable item name and it's value. In the backend servers, all the different addon preferences are stored using a MongoDB database. "A record in MongoDB is a document, which is a data structure composed of field and value pairs, which are similar to JSON objects"[cite] MongoDB is a NoSQL database and was used because of its similarity of storing JSON documents. NoSQL databases should mainly be used when the data stored on them is self-contained, is not relational and it is not needed to be doing JOINS between different documents.

Once the mirror has this information and everytime the user is identified by the FaceAPI, the mirror will update the addons with the latest user preferences and will initiate the addons loader. This specific piece of code is programmed to execute all the addons using the necessary commands. For example, an addon developed in Python would be started using "python addon_name.py", while one created using Node.JS

would be called using “node addon_name.js”. This addon loader can load addons from a wide variety of non-compiled programming languages and also compiled executables.

Once the user is happy with the desired configuration, it can be saved and, after a restart, the user is now ready to use the mirror with its updated configuration.

4.3 User interface

In the past, when people wanted to create desktop applications with graphical interfaces they had to use platform specific code and languages. Nowadays there are frameworks that allow the development of desktop apps using web technologies, such as “Chromium Embedded Framework”, “NW.js” or “Electron”. All of them make possible the creation of user interfaces and apps using the standard technologies in web development: HTML, CSS and Javascript.

The proposed application uses NW.js.

In Figure 4.5 it is possible to see how NW.js is an API layer on top of one Chromium browser engine and one Node.js instance. This works by connecting an HTML web page to a Javascript file which is in fact a Node.js file and so it has access to the Node.js API and functions. Since Node.js and Chromium are open source projects, both can be compiled to an ARM computer like the Raspberry Pi.

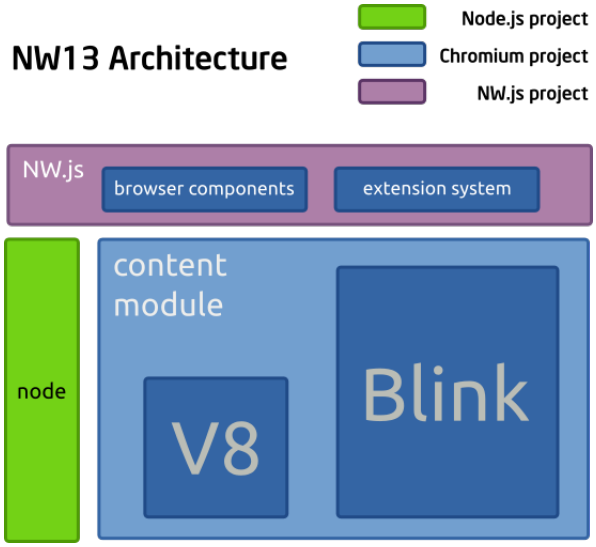


Figure 4.5: NW.js architecture

This type of device relies considerably on the user interface and user experience. The user can interact with two screens. The main screen and the specific viewer screen. When a user turns on the smart mirror, the main screen is the one being shown. In this screen, the addons can be displayed in five different spaces, those being the top left, the top right, the bottom left, the bottom right and the center of the screen.

In the first versions of the screen development, those spaces were supposed to represent different priorities in the addons. For example, more important addons would be standing in the middle of the screen while less important would stand on the bottom part. But that option received bad feedback and we made the decision to let users displays whatever addons they want wherever they want to.

From a programming point of view, the screens are HTML pages that have CSS styles and Node.js scripts associated to them. In the main screen, the five different regions are defined as HTML classes and each one of these classes have their style described in the main page CSS stylesheet. There, properties like position, margin, padding are set for each of those regions, that will hold the addons.

With only this screen available, addons developers would have to create applications that wouldn't occupy too much screen space and for some types of addons this condition would mean a bad user experience or an unusable addon. To break this restriction and allow developers to have full control over where the addons elements (text, images, ...) should be, the proposed solution offers a new screen, called internally, the viewer screen. From a programmer's perspective the HTML page for this viewer screen is just a centered inside the <body>. The addon can then use any style they want... in the end it will be put inside the . Addons that make use of this screen are initiated using an "open $addon" command where $addon is the name of the module we are trying to interact with.

To display addons to the mirror screen, the main application implemented in the "SmartMirror.js" file , is always listening on a specific message queue, and the addons enabled by the user, publish their "to-be presented" information on this queue, after they process the information needed for their job. All the addons use this same queue to publish their outputs which is then read asynchronously by the main application. For every new message that arrives at the queue, a function in the main application called "parseAddonMsg()" is invoked and starts the parsing of the message, which contains information about three things: the position of the screen in which the addon should go, the identifier of the addon and its output message in HTML format.

4.4 Speech interaction

At this point, a user of our system has his own customized screen continuously receiving information from the addons he choose. To enable interaction between the user and the smart mirror, the proposed solution uses speech recognition, providing the user with the means to use his voice to trigger actions upon the mirror information. The speech recognition module can be used by all the other addons that wish to use it.

When the main application is started, the speech recognizer is initiated in background and connects itself, as the only publisher, to the queue allocated to voice recognition. Unlike other queues in the application, this one uses a fan-out method where the speech recognizer is the only publisher and every add can subscribe and listen to this exchange by generating his own queue and associate it with the exchange, to receive the phrases pronounced by the user. We use this fan-out model because otherwise every addon would be connected to the same queue and once the speech recognizer publishes one message to the queue, only one addon would receive it, because it would be deleted afterwards.

Currently the proposed solution uses a mixture of offline recognition based on grammars, and cloud based speech recognition service, available from Microsoft Cognitive Services, mentioned in the state of the art chapter. Both speech recognizers work as separate background processes and are connected to the "voiceRec" exchange. The offline recognizer uses the Microsoft.Speech engine with the English language pack. The detection is based on grammars, that can be specified programmatically or by loading an XML file describing a grammar in SRGS format. A grammar is a collection of rules and each rule represents a word or phrase that the person using the mirror can speak to execute an action.

The grammar used on the offline recognizer expects an action keyword from the set: open, show, close followed by one word that is supposed to represent an installed addon. For example, "open videos" opens the video addon, but the recognizer will also understand "close" when the user intends to close an addon. Everytime the grammar detects a compatible word or phrase, the text is sent to the exchange. Any addon can then subscribe to the exchange and use the spoken phrases to provide new features and interaction to its' module.

Although this offline recognizer works really well and can understand with good accuracy what the user is speaking, we can't use it in cases where the user is not restrained to certain words or phrases, for example, when a user is supposed to search a song, a movie, a library of documents. In these cases, this recognizer doesn't understand what was said when the words are related to movie titles, music bands, book titles, and many other things. Another disadvantage is the fact that it was done in the C# programming language with the help of a .Net library, so it can only be compiled for the Windows platform.

To overcome these problems, the current solution uses the Bing Speech API, an online speech recognition service, available from Microsoft Cognitive Services. Unlike the previous offline solution, every phrase that a user says to the mirror is sent back to the Bing Speech servers, where it will be processed and converted to text, which is then sent to our application. Using this technique we obtained very good results and it is

now possible to search or take notes about anything. In spite of this, it is necessary to point out that there are two ways of accessing this service: through a REST API or using the C# client library. On one hand, using the REST API has the advantage of keeping our application multiplatform but on the other hand it has the disadvantage of being too slow for real time use, taking too long to send and receive the results from the servers.

Currently, and as a way to improve and establish the capabilities of our application, we decided to use the C# client library, which limits the application to being only used under the Windows platform, but allowing for quicker voice interaction between the system and the user, which we think it's a good trade-off.

Our common assumption is that most of the time, our system will be used intermittently and not so much in a continuous mode, where the user won't be constantly looking at the mirror, like what happens with current hardware-based personal assistants.

To inform the user in a better way, it was added a Text-To-Speech synthesizer function that every addon can use to "speak" to the user. The Chromium engine has a built-in TTS synthesizer and since NW.js runs on top of Chromium, we can allow the addons to use this function and convert text to speech. The main application is subscribed to the "voiceTTS" queue and when an addon publishes a message to this queue, the main application processes it and uses the synthesizer to produce the audio.

4.5 NOTIFICATIONS

Was created a way for addons to notify users when something meaningful happens. This can be done using "Toast" or more permanent notifications on the screen.

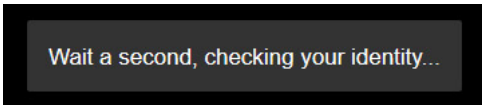


Figure 4.6: Example of a toast notification

A toast notification is a small and simple message with the intent of providing quick feedback to the user of the mirror. Addons and the main application can use this kind of notifications to provide visual confirmation that a task was finished, like for example, an e-mail was sent, or a to-do note was saved.

A more permanent notification can be used to make the user aware of a new content or updated information. These permanent notifications are messages that sit on the bottom left of the screen.

Once again, we use message queues to let any addon alert the user using these type of notifications. The main application is always listening in the "notify" message queue

where a function expects the type of message seen in the source code 2. An addon sends a message to this queue, where it specifies the message and if the notification is of the type TOAST or PERMANENT. Based on this type, the main application shows a toast message or adds the message to the bottom left of the main screen.

```
{
 "type":"TOAST|PERMANENT",
 "message":"This is a sample notification"
}
```

Code 2: Notification message

5

Modules

This chapter presents the different addons, that were developed to work with our proposed architecture and enhance the capabilities of the smart mirror.

5.1 Clock

One of the common features found in almost any electronic device nowadays and also present in all the other smart mirrors available is the digital clock. The digital clock in aggregation with his next calendar's events helps the user by making him aware of the current time, while he dresses or prepares himself for the day.

This was the first addon developed, because usually, programming languages make it easy to access the operating system's clock and a digital clock should be a simple addon that can help demonstrate our modular architecture.

The programming language selected for the development of the clock addon was Python. Python is an interpreted, high-level programming language, and allows most of the people who tries it to become quickly productive and produce or achieve results in short time.

To develop an addon in our architecture, the first thing we should do, is to create a connection to the message queue, that will be responsible for delivering our information to the main application. Since we are using RabbitMQ as our message broker, we use the "Pika" library, which is the recommended library from the RabbitMQ team, to communicate with the RabbitMQ service existent in our system.

Once we declare in which queues we want to publish and subscribe, we are ready to send data to them, and start the development of the addon itself.

Every addon has a configuration file and we start by reading those configurations. Our clock addon has currently no more configurations than the mandatory ones. But

in the future, could be added the option for a 12 hours clock instead of 24 hours. It could also be added the possibility for different date formats. In this case, to obtain the system's internal clock information we use a python module called "datetime", that lets the developer create "datetime" objects, which carry the information about the current date and time. As talked in the previous chapters, the main application expects a message to arrive in HTML format, so every minute, we create a new "datetime" object to obtain the updated date and time. With this information we create an HTML string with two paragraphs containing the updated time and date. As with most of the modules, we also add an icon to make the screen of the smart mirror more visually appealing. This icon is available from the FontAwesome collection.

The FontAwesome is a collection of free scalable vector icons created in CSS. The chosen icon for this addon was "fa-clock-o", which represents the clock icon, and is added to the previous HTML string by adding "". This HTML string is incorporated inside a JSON message containing the id of our addon ("clock") and the position (obtained from the configuration file) in which the addon will be shown on the screen.

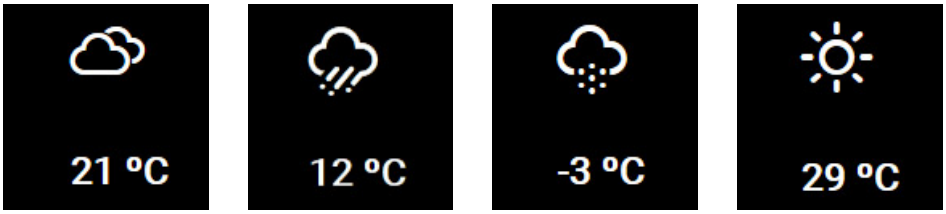
Figure 5.1: Clock addon.

5.2 Weather

Another common application in most smart mirrors and electronic devices is the weather application. In it's current state, our weather addon informs the user about the current temperature and weather status. It also serves as an addon example of how to interact with a public API. We use the OpenWeatherMap API to obtain free and updated weather data for the user specific city.

Unlike the previous addon, this second module was developed in Node.js, not for any particular reason, but to show that it is possible for the addons to be developed in a variety of languages. As before, the first step is to open a connection with and the "addons" queue. In Node.js we use the "AMQPLib" to interact with the RabbitMQ service. Once that is done, this addon starts reading the information from the configuration files, from where it obtains the user's city and the API key needed to retrieve OpenWeatherMap data. We consume the response from the OpenWeatherMap

in JSON format, mainly the fields about the current state of the weather and the current temperature and create an HTML string with that data. That HTML string is then used to create a JSON message and is sent to the main application. We use different AwesomeFont icons to represent a cloudy day, a sunny day, a rainy day and a snowy day. The different states of the weather addon are shown below, in figure 5.2.



(a) Cloudy (b) Rainy (c) Snowy (d) Sunny

Figure 5.2: Different states of the Weather addon

5.3 News

A smart mirror is a good device to receive quick updates about all things that might affect your day, such as the time, the weather or the news. The users of our system have available an addon that allows them to always be up-to-date with the latest news.

Microsoft Cognitive Services have a category of services dedicated to "Search", where we can find specific APIs to help the search of news, images, videos and text information on the web using Bing's search engine algorithms. In this application, we access the Bing News Search API to display trending news. It was programmed similary to the previous addons, using the Node.js language. After a period of time configured by the user, it consumes the data from the API and sends it to the main screen. Figure 5.3 shows the default appearence of the addon but the user may configure the number of news that are displayed at any given moment.

This addon uses voice recognition to provide two features. One of them is the possibility to reload the screen with other news. To activate it, the user should say "more news" and the current news on the screen are replaced with different news. Another feature that was implemented in this addon is the possibility for the mirror to narrate the news titles. This can be particularly useful while the user is getting ready for the day and isn't really paying attention to the smart mirror screen. This last feature is activated when the user says the words "read news"

This addon was developed using the Python programming language and one of the differences from the previous two addons is that, in order to use the voice recognition capability of the smart mirror, this addon is connected and always listening to the "voiceRec", where messages from the voice recognition system are published. Once it

receives a message with either “more news” or “read news”, the addon executes the command and refreshes the information on the main screen.

Figure 5.3: News addon.

5.4 Calendar

An additional application that is also very useful is the calendar. The calendar extension allows the user to have his Google Calendar synchronized on the smart mirror and have a more organized life and schedule. Figure 5.4 illustrates the state of this addon, while displaying, in the screen, the day and hours of a user’s event.

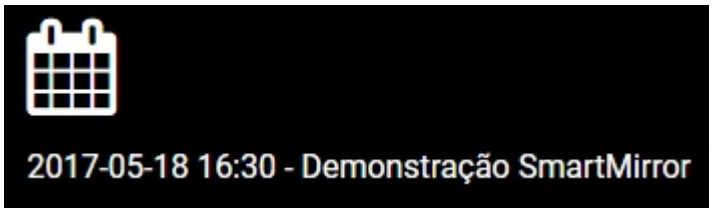


Figure 5.4: The Calendar addon displaying the user’s schedule.

We used the Node.js programming language to develop this module. Google wants developers to create applications that use their services and so, they provide various APIs for their services. One of them is the Google Calendar API, which we use in this addon, and gives developers the ability to modify and integrate Google Calendar data in their applications.

The Google Calendar API uses the OAuth2 protocol to authorize requests from applications to the Google Calendar data. So, for an application to use this data, first, the developer needs to register his application on the Google Developer Console and inform that that application will use the Google Calendar API. After this step, the developer receives a cliend ID and a client secret that will be used to identify the applications requests.

When our application requires access to the user data, our application asks Google for access while also notifying about the type of permissions needed, either read-only or read-and-write access. With this information, Google provides access to a web page where the user of our application can give our module those necessary permissions, saving a token to the local storage of the system. The next time our application requests data from the Google Calendar API, it will tie the saved token to the requests and we will have access to the information, if Google validates the request and the attached token.

Besides the display of the user's events, our Calendar addon allows the person using the smart mirror to add a new event to the calendar, using a voice command. A user can say "remind me" to initiate the construction of a new event. The mirror will then ask "What should I remind you of" to which the user can respond with any phrase, and this response will be saved on the calendar as a daily event.

Currently, the data obtained from this addon is only obtainable from a test account that was already configured to allow both read and write access to its calendar data. Because of previous decisions related to both the architecture behind the addon's configurations and the development made in the configuration page, the proposed solution doesn't possess, currently, a way to give a new user, access to the consent screen where the user should login into his Google account and authorize the access to his calendar information.

Future work should be made towards the creation of a new section inside the configuration page, where the user is able to link his various accounts, for example, his Facebook, Twitter, Google account and authorize, in that page, access to the information obtainable through those accounts and services.

5.5 Welcomer

As already declared before, the system is turned on to show a default screen with only two addons. When the user is in front of our mirror and is interested in having access to more than the default page, he can login to his page, previously configured, using the voice command "hello".

When the mirror listens to this command it will execute the identification of the user, using the face recognition service at Microsoft Cognitive Services. Since we previously created a new account and uploaded one facial image to the Face API service, we are now able to take a photo of the user in front of the mirror and ask the Microsoft Cognitive Services if they can recognize and identify the person in that photo. Since the implementation of this recognition step, this Cognitive service always worked very well, identifying the correct user nearly everytime.

If the person is recognized by the service, the email associated with the person is received by the smart mirror and the smart mirror now knows who is the current user and reloads the information on the screen to reflect the state and configurations saved by the user.

Another alternative to the smart mirror react to persons and users in front of it consists in endowing movement sensors to our smart mirror and, as seen in figure 5.5. When not in use, the smart mirror would be in a standby state and would wake up to the default screen when the movement sensors detected movement and possible users in front of it. In that instant the smart mirror would try to recognize the user and, if successful, would show him his screen.

Because we didn't had access to physical movement sensors, an experiment using video processing was made. Using the functions available in the OpenCV library, we made an attempt at creating our own movement detector using algorithms based on background subtraction, specifically, the MOG2[25][26] algorithm.

The resulting program tried to recognize the person in front of the mirror everytime the background observed changed drastically and failed most of the times because when the photo was taken, the person was not completely looking into the webcam, or because the image taken much before the webcam could see all of the person's face. Because of this, the proposed mirror used first solution described and only tries to recognize the user once the user intends to be recognized and says the voice command "hello". This background subtraction work could still be used to wake up the mirror screen from a sleep state, from a turn off monitor to the default screen.

We think that electronic devices like the smart mirror, which can have a great impact in our daily lives, are more interesting when they can understand all the context in which they are communicating with the user. The proposed solution incorporates the Emotion API, from Microsoft Cognitive Services, into our face recognizer function to explore this concept and give the system a more natural interaction with it's user. The Emotion API allows an application to be able to recognize a person's emotion based on a picture of her face.

At this time, we only use this service to make the smart mirror system seem more personal, giving it the ability to ask the user if everything is okay when it detects a negative emotion like sadness or fear, or show a funny image when the user is happy for example. An example of these kind of messages can be seen in Figure 5.6.

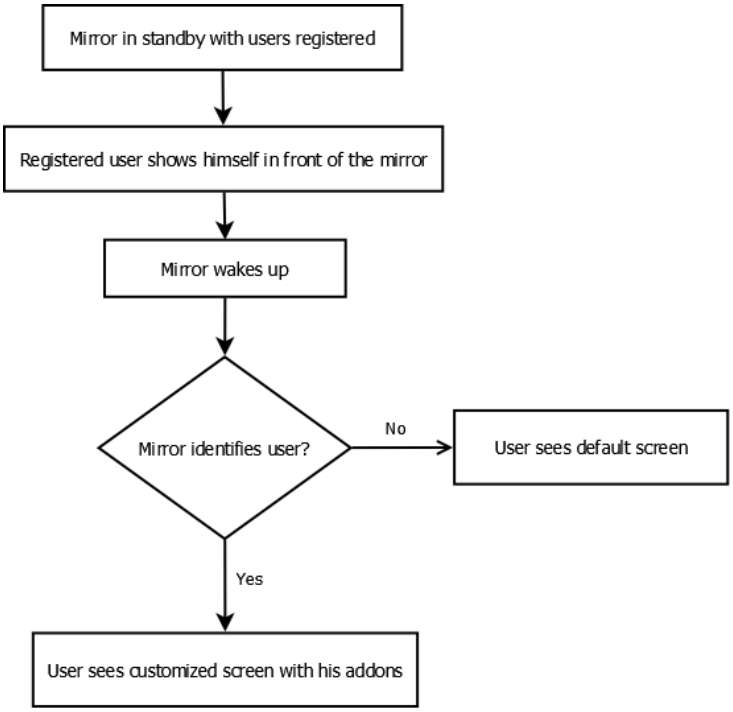


Figure 5.5: Diagram representing an alternative approach to our current recognition system.

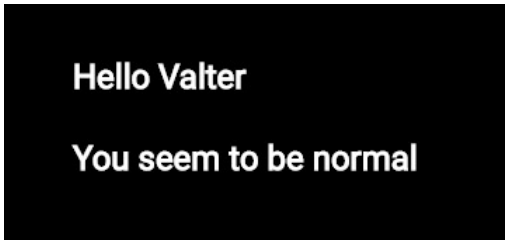


Figure 5.6: A welcome message is presented when the user is recognized.

5.6 Fun glasses

Augmented reality games and applications are something that has existed for a long time but are now getting really used since the famous Snapchat, Facebook and Instagram started using it to create their photo filters in real time, where it is possible for a person to look like a dog, a cat and many other animals. Since our device is a mirror and people use mirrors mainly to try out different clothes and accessories, we created a simple augmented reality application where the user is capable of trying new glasses and ties.

To develop this application, we looked at Python and the OpenCV library. The OpenCV library is a collection of useful functions that are essential for working and manipulating images and video frames.

To be able to augment a person's face with glasses or other accessories, we first need

to find the face area in the image and retrieve points of interest from that area, such as, the pixels in which the eyes are situated, the pixels that delimit the face and so on. This task becomes easier with a library like DLib[22], that is able to detect sixty-eight of these landmark points, when analysing a picture containing a person's face. So after taking a photo of the user and processing it using the DLib functions, we have many points of interest from the user's face and we now want, for example, to place a pair of glasses on his face, so we start by resizing the glasses image such that it has the same width as the person's face and place it in the exact middle between the person's eyes.

At this point we have a rectangle area of pixels where we are going to paint the glasses. The glasses image is saved in the PNG file format with a transparent layer and the task of painting the actual glasses over the person's face becomes as simple as replacing the pixels from the person's face with the pixels from the glasses picture, only if the pixel in the glasses image is not transparent.

The same principle is applied when we want to place a tie on the user's body, except in this situation we rely on the chin landmark point.

The final image is then converted to a base64 string and send through the addons queue to be displayed on the mirror screen. Figure 5.7 pictures the way is image in seen on the mirror, while figure 5.8

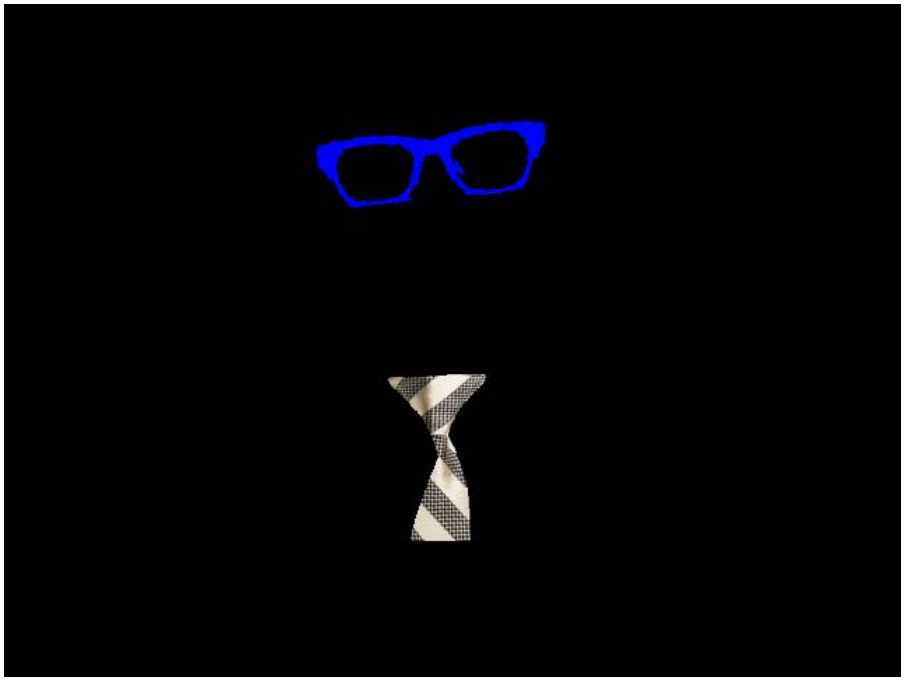


Figure 5.7: Glasses and tie as seen from the mirror.

Figure 5.8: Result of addon fun glasses on the mirror

5.7 Restaurant meals

The proposed solution has available an addon in charge of informing the user about the daily meals menus for both the Castro and Santiago canteen as well as the restaurant located in the University of Aveiro. The weekly menus are made available by this institution through a free and public API. This addon uses Node.js to consume the data and belongs to the category of addons that are shown in the additional "viewer" screen.

When the user runs this addon, the user will be able to see the next meal menu for the canteen of Santiago, so if the user is opening the addon before the 14:00h hour mark the menu shown will be in respect to the lunch period and otherwise will be shown the menu for the dinner period.

A user can navigate through the different menus and canteens using voice commands, that are recognized by this module. Those are:

- **lunch** displays the lunch menu;
- **dinner** displays the dinner menu;
- **next** displays the next menu on the next possible meal place;
- **previous** displays the previous menu.

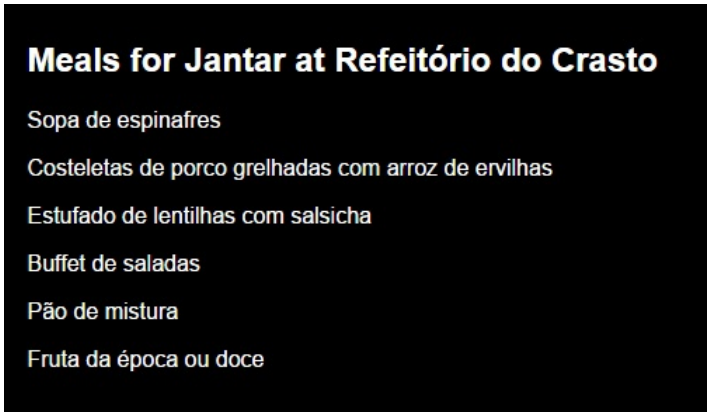


Figure 5.9: Meals addon.

5.8 Videos

Being able to play a video in the smart mirror can have many advantages. A woman can use it to play a video of a new make-up tutorial while she is trying it on herself. A user with a smart mirror on the kitchen can play a new recipe video to help him prepare a new meal. A user can always use it to play some music videos or comedy videos to better enjoy is time in front of the mirror, be it in the morning or in the evening.

When we talk about online videos and video streaming websites, one of the first platforms that comes to mind is definitely Youtube, so this addon allows the user to watch videos from Youtube. The addon in question was programmed using the Node.js programming language with the help of third party library "youtube-node" to search and access the content of the Youtube platform.

Because this is a more complex addon, where we need to display various images and information, it makes perfect sense to use the "viewer" screen once again for displaying both components of this addon, which consist of a page with the search results (Figure 5.10) and the a screen where the video is playing (Figure 5.11).

When the application is initiated, we see a message telling the user that he should search for a video to be able to play one. In the future, this initial page can show previously seen but not finished videos, or more videos recommended based on the user's interests.

The user interaction is made through voice recognition right from the start, because, in the current state of the module, the user can only play a video after he searches for one, and he does it using his voice. So, the user can search videos by using the command "search" followed by his search terms. For example, "search john oliver" will search the Youtube platform for videos related to "john oliver". Once the search query is understood by the module, four search results will be shown to the user, and we can see an example of this represented in figure 5.10.

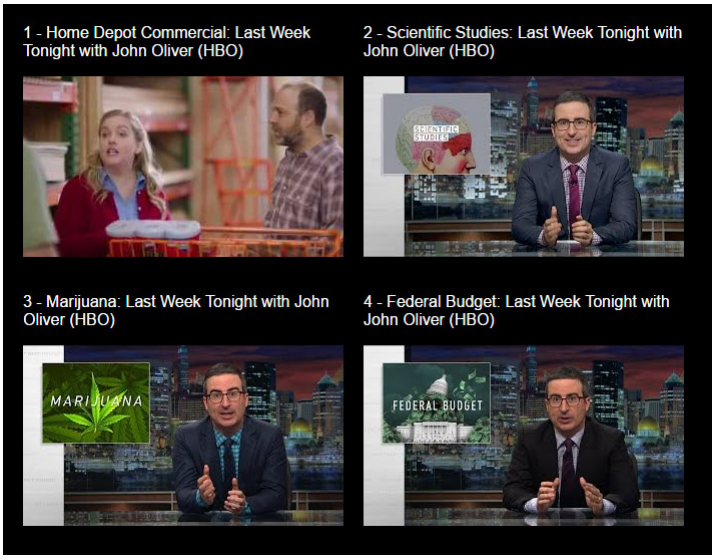


Figure 5.10: Search results on the Videos addon, for the query "john oliver".

Figure 5.11: Video player.

The figure displays four search results in the form of video thumbnails as well as the video title and number of the video on the list. Based on this number, the user can then use another voice commands such as "play video number one" or "play first video" to play the desired video. The video will then start playing and in a new screen. In there, the user as the option to go back to the search results screen using the voice command "back" or exit to the main screen by saying "close".

As a side note, it should be stated that some videos can't be played on the smart mirror because we embed the video in our page, and those videos can only be played through the Youtube website itself.

5.9 Podcast

With some people already testing the prototype and with most of the interaction already built, we received the idea to build a podcast addon, shown in Figure 5.12, that would enable the user to listen to his favorites podcasts while using the smart mirror.

The podcast module was created using Node.js and uses a third-party library, "xml2js", to transform the data obtained, in the XML format, from the RSS feed into object usable Javascript objects, which permits a faster implementation of the logic associated with the addon.

To use this addon, the user should, foremost, configure his favorite podcast series by updating the link of the RSS feed in the configuration page. Normally this would mean that only one series at a time could be loaded in the addon, but the user can use a website such as RSSMix [27] to combine various RSS feeds into only one and then configure the addon to use it.

Once the addon is ready and the mirror is initiated, the list of episodes in the configured podcast is loaded in memory and the user can advance on this list by using the voice command "more podcasts", causing a refresh to the displayed podcast. At this point, if the user is interested in the episode, the user can use the voice command "play podcast" to start the playback of the episode. This is achieved by using the HTML "<audio>" tag with the source url modified to the one obtained from the RSS feed data.

The user can stop the playback of the episode at any given time using the command "stop podcast". Internally, for this to happen, the addon sends a new message but with without the audio tag and so the HTML code on the main screen is updated with this new message and the podcast is stopped.

Figure 5.12: Podcast addon.

6

Experimental Studies

This chapter reveals the experimental testing done to the current prototype and discusses the feedback and interactions made by the users.

With the purpose of testing the quality of our prototype and understanding the opinion of the other persons towards our design, an experimental study was made with the help of various subjects. At this stage it is important to answer some questions, such as:

- Can the user correctly register a new account?
- Does the user understand how to add new addons?
- Can the user figure out how to configure the addons?
- How good is the interaction with the mirror using voice commands?
- Would the user adopt this kind of device?

We asked twelve software engineers from the Altice Labs company to test our system, independently and one by one, and observed their behaviour while interacting with our prototype. Before the user started using our product, he was told that he just bought that smart mirror and finished installing it in his house and is now ready to be configured using the configuration page, which is available at the devices's IP address using port 8080.

The person was then asked to access this configuration page, create a new account for him to use with the smart mirror, include and configure some addons and finally, to interact with the smart mirror anyway he would see fit. Through all of this contact with the system, some notes, that help respond the stated questions, were taken. Table 6.1 summarizes those notes.

Observed behaviour	Yes	No
Able to register an account properly	0	12
Understood how to add addons	9	3
Tried to change the configurations	6	6
Interacted with addons without external help	3	9

Table 6.1: Results obtained from behaviours tests

The observations made during the various tests show that all of the users had problems creating a new account, specially when taking the photo that is used in the Face API, which means this registration page should be modified to improve and allow for a better user experience. This conclusion differs from the results obtained in the first question of the questionnaire6.1, that was made to the users. Eleven out of the twelve testers thought that the process of making a new account was easy or somewhat easy.

Part of the attention, while the user was testing our system, was given to how easily could the user understand that the addons are added to the system by drag and dropping them from the "Addons available" section to the desired position in the screen. Ten of our testers understood how to correctly add addons to the smart mirror without any help but two persons needed a tip on what was supposed to do on the screen represented in figure 4.4. Although, we didn't had a broad number of people testing our system, these results help our supposition that using drag and drop is a good way of installing and configuring addons in the system. The second question in our survey addresses this issue, and eleven people responded that changing the addons position in this way was easy.

Another important part of this test was to perceive if the users realized how they can modify the configurations of the addons. Exactly half of our study population understood how to do it and proceeded to modify their configurations to their preferences, while the other half had a bit of trouble with that task.

The third question of our questionnaire asks users, on a scale of one to five, how difficult was the process of configurating new addons. To this question, we had four users responding with a three, meaning not difficult but not easy also, another four users responding with a four, which can be interpreted as a process being understandable but could use some improvement and the other four answered with a five, meaning that they thought modifying the addons configurations was easy. These results are in agreement to our perception of the test.

Judging how well a new user of our smart mirror can interact with the addons was also an essential factor in this small study. From our observations, nine out of the twelve

testers needed help when interacting with the smart mirror using the voice commands even though they knew about the "help" command which shows all the possible voice commands. The other three were able to interact quite well with the addons using the help mechanism provided by the system. The result of this observation should not be that strange, when considering the fact that all the users were using and interacting with the smart mirror for the first time and most of them installed many addons on their systems so there were many voice commands available from the start, which can put the user in a position where he can't absorb all that information at once.

Our questionnaire approaches this subject and asks the user "How do you classify the voice interaction with the system?" and "Did you find the help provided useful?". The first question received a three from the majority of the testers, meaning they consider the interaction to not be hard but also not easy. Regarding the second question, "useful" and "very useful" were the preferred options with a combined choice from ten users. In addition to this, almost all the users, mention in the open commentary section of the questionnaire that they would like for these help alerts to be continuous and always visible on screen, at least during the first times of interaction.

The last question asked to the user was "Would you like to have this kind of device in your house?" and it received all kind of responses. Two persons responded that they would not like to have this device in their homes and one user also responded that he may not like to have one smart mirror. Another user responded with "maybe, not sure" and the rest of the users gave a positive answer by responding that either probably or for sure like to have a smart mirror in their homes.

As a last note, is useful to state that we also looked at the behaviour of the face recognizer used. It was able to recognize correctly ten out of the twelve users. In one of the situations, it failed because the user took a bad photo of his face (cropped part of it) when creating his account. We couldn't understand the reason for the other failure but it could be that the user also had a photo with a bad (for the system) facial expression or his glasses could be causing problems with the recognizer, although other persons had glasses and it worked correctly for them.

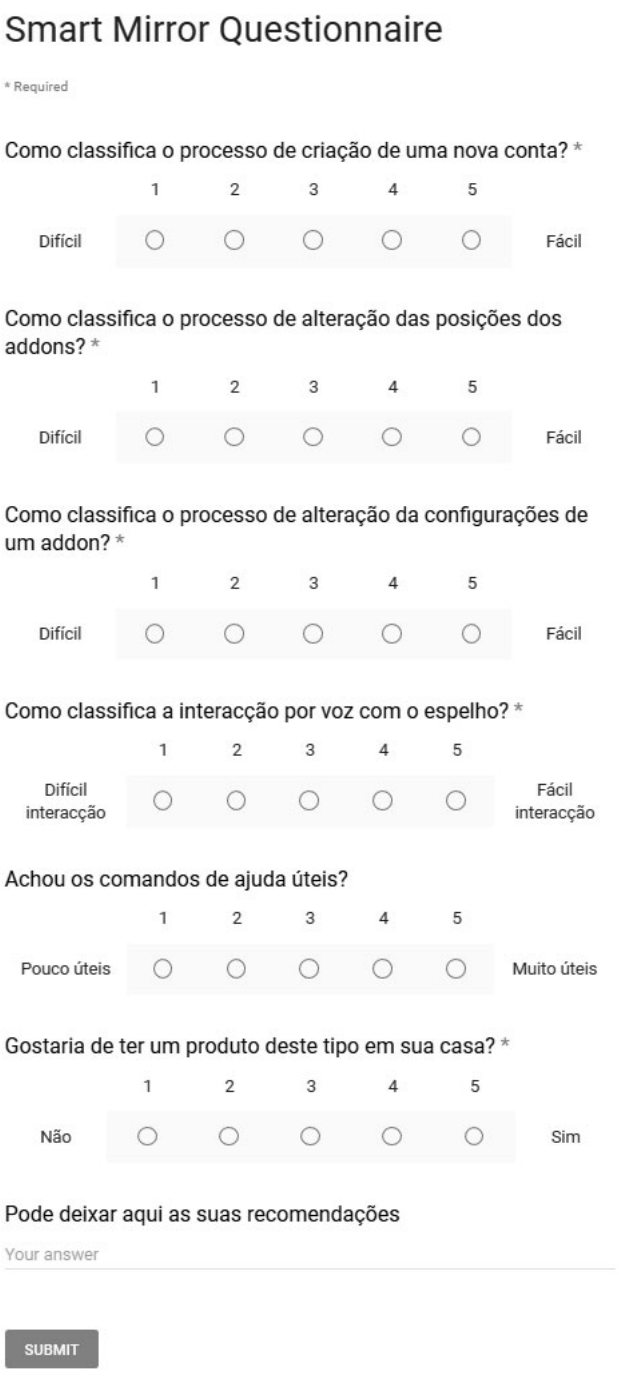


Figure 6.1: Questionnaire made after the tests

7

Conclusion

This chapter exposes the conclusions and problems encountered, while also discussing the future work that may be done in order to improve the project.

To summarize, an interactive and modular smart mirror was created. Several addons were also developed to demonstrate the capabilities of the system and the system has now an interesting base that allows other developers to create design and add more extensions to it. This system was tested and assessed by several people using a real prototype, where it was possible to understand the system's best and worst characteristics.

The proposed application has the advantage of allowing the development of modules in any language that the developer considers adequate, provided that the programming language can communicate with the message queues using some tools or libraries, and an example of that is the use of Python, which grants an easier access to the webcam and image transformation libraries like OpenCV.

Initially and like all of the current smart mirrors, an objective for this system was to make it cross-platform, so that it could run in any computer and operating system, but in order to improve the final prototype some trades were made. One of them has to do with voice interaction where a C# client library is used to send and consume data from the Bing Speech API. This client library is provided by Microsoft and at the time of writing was only available for the Windows platform but by using it we achieve quicker voice recognition and better interaction with the mirror. For this reason, the final prototype is not multiplatform and thus cannot be fully installed in a mini computer like the Raspberry Pi running Linux, which most of the smart mirrors use nowadays. But that can be achieved by modifying the voice recognition module and use the REST protocol to access the Bing Speech API.

Although the proposed solution is not yet ready for mass production and usage by the average consumer, all the work done is seen as essential by Altice Labs, as it provides them an initial step in understanding the different problems and concerns that may arise when creating and manufacturing this kind of device. The present prototype provides the base for various testing. Now, Altice Labs has the possibility to test new approaches, for example, in the privacy component or the way users interact with the system, that might include not only voice, but also hand gestures or physical touch buttons, for instance. Was also created the foundation for the development of new addons, as suggested in the next section.

We are very grateful for the opportunity to collaborate with Altice Labs in investigating, designing and bringing to life a new smart mirror, that tries to improve the current state of the art. This could be a device that might gain a large attraction in interest from end-users and might be someday seen in the majority of households and Altice Labs is already a step closer in contributing that future.

7.1 Future Work

There are many ways in which the presented work can be improved.

The proposed system uses the recognition of voice commands to allow the interaction between the user and the smart mirror but, as stated in the state of the art chapter, this interaction could be made through gestures recognition, or using a touchscreen, (or) maybe an application on the smartphone, or perhaps the product could provide physical buttons that the user could use.

Another path for improvement is to develop new modules such as:

- Music players, which can obtain their data from Meo Music, Spotify, Pandora or simply from the product internal storage.
- Live television player which might use the work done at MeoGo.
- Health modules, which may harvest information from different health and fitness applications and devices (smartwatches or fitness bands) with the intent of providing users with a healthier lifestyle.
- Social media addons which can keep the user notified about their latest interactions in social media platforms such as Twitter, Facebook, Instagram, Snapchat, Linkedin or Quora.
- Addons that can communicate with IoT devices and allow interaction with window's blinders, air conditioning, light bulbs and many more.
- Modules created to engage the user with the smart mirror and contribute to a funnier interaction such as allowing the user to try more pieces of clothing, a game while a user is brushing his teeth or a module for the user to take photos of himself.

Advances in the system may also be achieved by creating a new way of specifying the style of the addons because currently all the style information is being sent through the message queue every time the addon is updated and better performance could be achieved if the styling was already loaded in advance and the addon only had to update some values. Some improvements could also be made to the way addons are stored in the internal storage because currently they are being stored at an user level and could be stored at a system level, like a modern operating that has applications installed which can be used by various users. The method that addons use to interact with local configurations could also be expanded to allow for more diversity and the existence of global and local configurations.

References

[1] *Altice labs website*, http://www.alticelabs.com/en/, [Online; accessed 07-06-2017].

[2] K. Fujinami, F. Kawsar, and T. Nakajima, "Awaremirror: A personalized display using a mirror", in *International Conference on Pervasive Computing*, Springer, 2005, pp. 315–332.

[3] M. A. Hossain, P. K. Atrey, and A. El Saddik, "Smart mirror for ambient home environment", in *Intelligent Environments, 2007. IE 07. 3rd IET International Conference on*, IET, 2007, pp. 589–596.

[4] *Samsung mirror*, http://displaysolutions.samsung.com/digital-signage/detail/848/ML55E, [Online; accessed 09-01-2017].

[5] *Lg mirror*, http://www.lg.com/au/commercial/mirror-signage/lg-49MS75A, [Online; accessed 09-01-2017].

[6] *Toshiba mirror*, http://newatlas.com/toshiba-smart-mirror-concept-ces-2014/30574, [Online; accessed 27-01-2017].

[7] *Perseus mirror website*, http://www.perseusmirrors.com/, [Online; accessed 28-11-2016].

[8] *Perseus teaser video*, https://www.youtube.com/watch?v=yNXwBszZyQQ, [Online; accessed 14-07-2017].

[9] *Magicmirror website*, https://magicmirror.builders/, [Online; accessed 16-11-2016].

[10] *Homemirror website*, https://github.com/HannahMitt/HomeMirror, [Online; accessed 16-11-2016].

[11] D. Gold, D. Sollinger, *et al.*, "Smartreflect: A modular smart mirror application platform", in *Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual*, IEEE, 2016, pp. 1–7.

[12] *Duo | an ai computer for your home*, https://duo.computer/, [Online; accessed 19-05-2017].

[13] D. Schmidt, F. Block, and H. Gellersen, "A comparison of direct and indirect multi-touch input for large surfaces", in *IFIP Conference on Human-Computer Interaction*, Springer, 2009, pp. 582–594.

[14] I. AlAgha, A. Hatch, L. Ma, and L. Burd, "Towards a teacher-centric approach for multi-touch surfaces in classrooms", in *ACM International Conference on Interactive Tabletops and Surfaces*, ACM, 2010, pp. 187–196.

[15] A. Schick, F. van de Camp, J. Ijsselmuiden, and R. Stiefelhagen, "Extending touch: Towards interaction with large-scale surfaces", in *Proceedings of the ACM international conference on interactive tabletops and surfaces*, ACM, 2009, pp. 117–124.

[16] E. W. Pedersen and K. Hornbæk, "An experimental comparison of touch interaction on vertical and horizontal surfaces", in *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, ACM, 2012, pp. 370–379.

[17] U. Von Zadow, W. Büschel, R. Langner, and R. Dachselt, "Sleed: Using a sleeve display to interact with touch-sensitive display walls", in *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, ACM, 2014, pp. 129–138.

[18] S. Boring, D. Baur, A. Butz, S. Gustafson, and P. Baudisch, "Touch projector: Mobile interaction through video", in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2010, pp. 2287–2296.

[19] *Kairos website*, `https://www.kairos.com/products`, [Online; accessed 25-01-2017].

[20] *Face++ website*, `https://www.kairos.com/products`, [Online; accessed 25-01-2017].

[21] *Microsoft cognitive services website*, `https://www.microsoft.com/cognitive-services/en-us/`, [Online; accessed 15-12-2016].

[22] *Dlib website*, `http://dlib.net/`, [Online; accessed 15-12-2016].

[23] *Cmu sphinx website*, `http://cmusphinx.sourceforge.net/`, [Online; accessed 27-01-2017].

[24] *Opencv website*, `http://opencv.org/`, [Online; accessed 30-05-2017].

[25] Z. Zivkovic, "Improved adaptive gaussian mixture model for background subtraction", in *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, IEEE, vol. 2, 2004, pp. 28–31.

[26] Z. Zivkovic and F. Van Der Heijden, "Efficient adaptive density estimation per image pixel for the task of background subtraction", *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.

[27] *Rssmix website*, `http://www.rssmix.com/`, [Online; accessed 08-07-2017].

Appendix

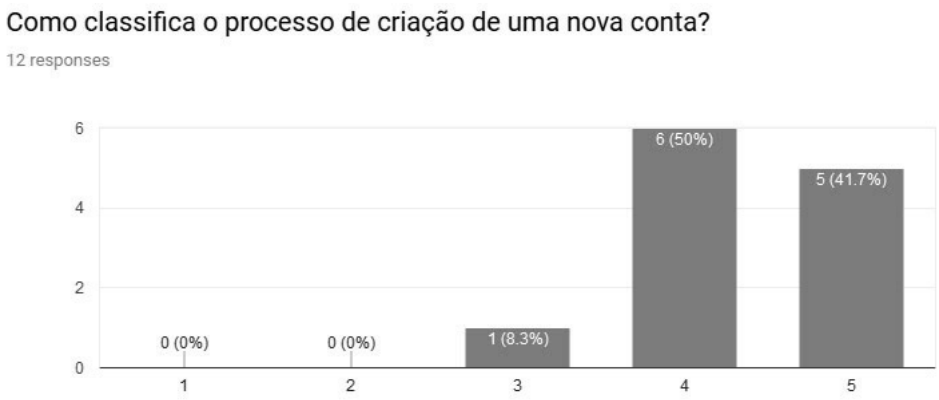


Figure 1: Results to the first question of the questionnaire

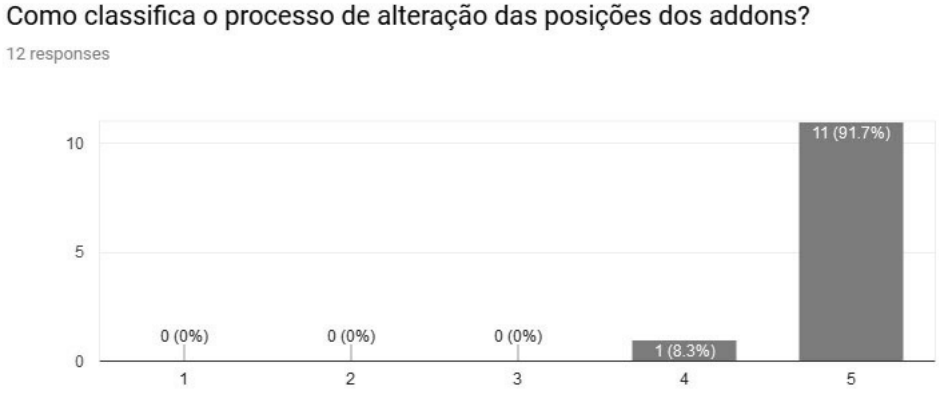


Figure 2: Results to the second question of the questionnaire

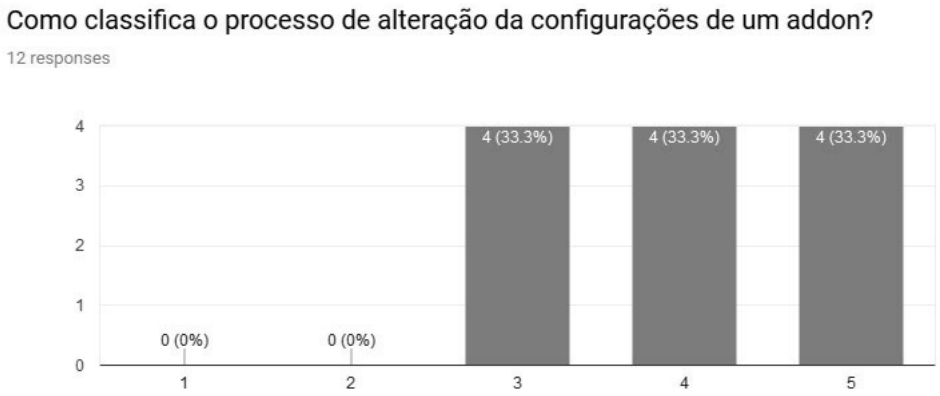


Figure 3: Results to the third question of the questionnaire

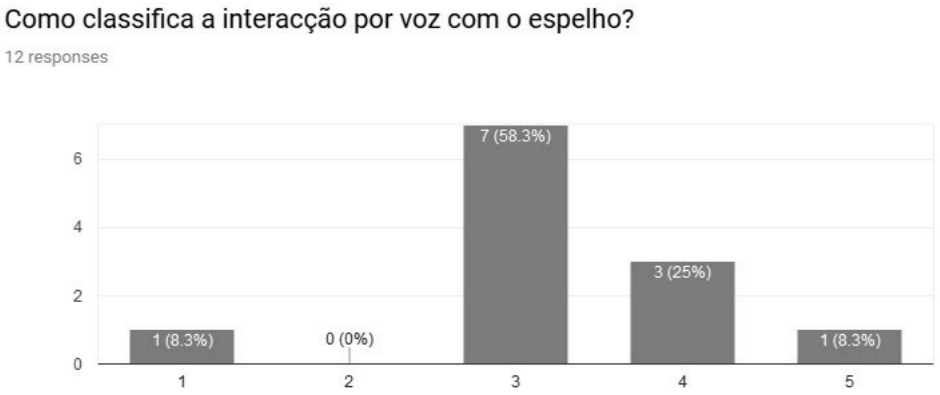


Figure 4: Results to the fourth question of the questionnaire

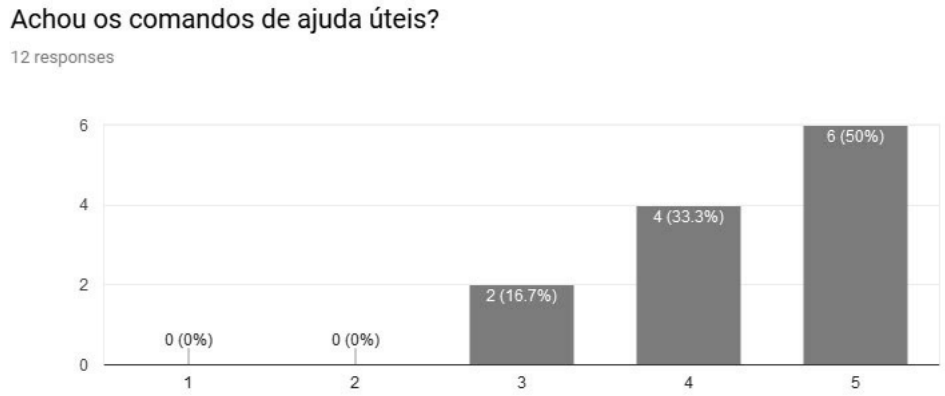


Figure 5: Results to the fifth question of the questionnaire

Figure 6: Results to the sixth question of the questionnaire